IT & C

ISSN 2821 - 8469, ISSN – L 2821 - 8469, Volumul 1, Numărul 1, Septembrie 2022

Tipuri de date atomice în Python

Stephen Davies

Pentru a cita acest articol: Davies, Stephen (2022), Tipuri de date atomice în Python, *IT & C*, **1**:1, 32-41, https://www.internetmobile.ro/tipuri-de-date-atomice-in-python/. Traducere și adaptare independente Nicolae Sfetcu

Publicat online: 21.08.2022

ABONARE

Tipuri de date atomice în Python

Stephen Davies

Rezumat

Când spunem că unele date sunt „**atomice**", nu înseamnă că sunt radioactive; vrem să spunem că sunt *indivizibile*.

Anticii vorbeau despre „atomi" drept cele mai mici bucăți posibile de materie. Dacă împărțiți orice obiect fizic – să zicem, un măr – în părți, obțineți componentele sale: o tulpină, pielița, semințe și miezul dulce și suculent. Tăiați oricare dintre aceste bucăți cu un cuțit și veți obține bucăți mai mici. Dacă continuați să le divizați din ce în ce mai mult, filozofii precum Democrit au argumentat că veți ajunge în cele din urmă la mici biți indivizibili care nu mai pot fi divizați în continuare. Aici se află lumea fizică la cel mai înalt grad de granularitate.

În mod similar, o piesă de date atomice este tratată de obicei ca o unitate întreagă, nu ca ceva cu structură internă care poate fi defalcată. Există diferite moduri în care acești atomi de date pot fi strânși împreună și organizați în ansambluri mai mari.

Sursa: Stephen Davies, *The Crystal Ball – Instruction Manual*, Vol. 1: Introduction to Data Science, v. 1.1. Copyright © 2021 Stephen Davies. Licența CC BY-SA 4.0. Traducere și adaptare independente: Nicolae Sfetcu

Cuvinte cheie: date atomice, Python

IT & C, Volumul 1, Numărul 1, Septembrie 2022, pp. 32-41
ISSN 2821 - 8469, ISSN – L 2821 - 8469
URL: https://www.internetmobile.ro/tipuri-de-date-atomice-in-python/



Date atomice

Când spunem că unele date sunt „**atomice**”, nu înseamnă că sunt radioactive; vrem să spunem că sunt *indivizibile*.

Anticii vorbeau despre „atomi” drept cele mai mici bucăți posibile de materie. Dacă împărțiți orice obiect fizic – să zicem, un măr – în părți, obțineți componentele sale: o tulpină, pielița, semințe și miezul dulce și suculent. Tăiați oricare dintre aceste bucăți cu un cuțit și veți obține bucăți mai mici. Dacă continuați să le divizați din ce în ce mai mult, filozofii precum Democrit au argumentat că veți ajunge în cele din urmă la mici biți indivizibili care nu mai pot fi divizați în continuare. Aici se află lumea fizică la cel mai înalt grad de granularitate.

În mod similar, o piesă de date atomice este tratată de obicei ca o unitate întreagă, nu ca ceva cu structură internă care poate fi defalcată. Există diferite moduri în care acești atomi de date pot fi strânși împreună și organizați în ansambluri mai mari.

Medii și variabile

Un program de analiză a datelor folosește un **mediu** pe măsură ce rulează. „Mediu” înseamnă doar „toate datele care sunt vizualizate în prezent și pe care programul le poate accesa.” (1) Mediul este format din **variabile**, fiecare (de obicei) având un **nume** și o **valoare**. De exemplu, s-ar putea să avem o variabilă numită vârstă a cărei valoare este 21 și o variabilă numită `slogan` a cărei valoare este „`Unu pentru toți și toți pentru unu`”.

Fiecare variabilă din mediu trebuie să aibă un nume *distinct* (adică, nu există două variabile care să poarte același nume). De asemenea, important, motivul pentru care aceste elemente de bază sunt numite „variabile” este că valoarea lor se poate *schimba* pe măsură ce programul se execută.

Deși putem crea inițial o variabilă **age** cu valoarea 21, mai târziu în program, valoarea variabilei s-ar putea schimba la 22, sau 50 sau 0. *Numele* variabilei nu se schimbă niciodată.

Tipuri de date atomice

Există un alt lucru pe care o variabilă îl are pe lângă numele și valoarea sa: un **tip**. (2) Într-un limbaj de programare precum Python, fiecare bucată de date are un tip specific, care este necesar pentru a determina cum se comportă și tot ce puteți face cu ea. O întrebare pe care ar trebui să o pui mult este: „bine, am o variabilă în mediul meu numită **x** … acum care este tipul ei?” Este posibil să fi ghicit (corect) că variabilele noastre de **vârstă** și **slogan** din secțiunea anterioară sunt de diferite tipuri: una este un număr, iar cealaltă este o frază.

Există trei tipuri principale de date atomice.

Numere întregi

Un tip foarte comun de date sunt numerele întregi sau întregii. Acestea sunt de obicei pozitive, dar pot fi și negative, și nu au nicio zecimală. Lucruri precum anul nașterii unei persoane, votul total al unui candidat sau numărul de „aprecieri” al unei postări pe rețelele sociale sunt reprezentate cu acest tip de date.

Numere reale (fracționale)

Vă puteți aminti din matematica liceului că așa-numitele „numere reale” includ nu numai numere întregi, ci și numere cu cifre după virgula zecimală. Prin urmare, acest tip poate fi utilizat pentru a stoca ratele dobânzilor, citirile de temperatură și ratingurile medii ale filmelor pe o scară de la 1 la 5.

Deoarece toate numerele întregi sunt numere reale, s-ar putea să vă întrebați de ce ne deranjăm să definim două tipuri diferite pentru acestea. De ce să nu dai doar ambelor tipuri de variabile același tip, de număr real? Practic, răspunsul este că ceva „se simte a fi greșit” în acest sens pentru comunitatea științei datelor. Un utilizator Facebook ar putea avea 240 de prieteni sau 241, dar niciodată nu ar avea sens să aibă 240,3 prieteni. A apărut astfel un consens: variabilele care ar stoca numai numere întregi ar trebui să fie într-adevăr de un tip dedicat doar numerelor întregi. Puteți încălca această convenție, dar veți fi considerat ciudat de colegii dvs. dezvoltatori dacă faceți acest lucru.

Text

În cele din urmă, unele valori care nu sunt deloc numerice, cum ar fi numele unui client, titlul emisiunii sau un tweet. Deci, al treilea tip de date este textual. Variabilele de acest tip au o secvență de caractere ca valori. Aceste caractere sunt de cele mai multe ori litere, dar pot include și spații, punctuație și caractere din alte alfabete.

Apropo, acest al treilea tip de date se poate îndrepta până la linia „atomică" și uneori o poate traversa. Cu alte cuvinte, vom lucra ocazional cu valorile textului non-atomic, împărțindu-le în cuvintele lor constitutive sau chiar în litere. De cele mai multe ori, totuși, vom trata o secvență de caractere precum „Avengers: Endgame" ca o singură bucată de date indivizibilă în același mod în care tratăm un număr ca 42.

Dar ce zici de …?

Ce se întâmplă cu alte lucruri pe care le poate stoca un computer: fișiere de imagini, melodii, videoclipuri? Se pare că, prin trucuri inteligente, toate aceste tipuri de suporturi și multe altele pot fi reduse la un număr mare de numere întregi, și stocate într-o structură de date cumulată. La nivel atomic, rămân cele trei tipuri prezentate mai sus.

Cele trei tipuri în Python

Cele trei tipuri obișnuite de date atomice descrise mai sus sunt în **limbaj general**: aceasta înseamnă că sunt conceptuale, nu sunt legate de niciun limbaj de programare specific sau instrument de analiză. *Orice* tehnologie utilizată pentru știința datelor va avea capacitatea de a face față acestor trei tipuri de bază. Modalitățile specifice în care fac acest lucru vor diferi oarecum de la o limbă la alta. Să aflăm cum le implementează Python.

Numere întregi: int

Unul dintre cele mai de bază tipuri de date Python este „**int**", care înseamnă „întreg". Este ceea ce folosim pentru a reprezenta numerele întregi.

În Python, creați o variabilă prin simpla tastare a numelui său, a unui semn egal și apoi a valorii sale inițiale, astfel:

```
revolution = 1776
```

Aceasta este prima noastră **linie de cod** (3). După cum vom vedea, liniile de cod sunt **executate** una câte una – există un timp înainte și un timp după, fiecare linie este efectiv executată.

Acest lucru se va dovedi a fi foarte important. (Oh, iar o „linie de cod” este uneori numită și o **instrucțiune**.)

Numele variabilelor Python pot fi cât de lungi doriți, cu condiția să fie alcătuite doar din litere mari și mici, cifre și caractere de subliniere. (Trebuie să fiți în concordanță cu scrierea dvs. cu majuscule și cu ortografiia dvs.: nu puteți apela o variabilă **Movie** într-o linie de cod și **movie** în alta.) Sublinierile (undersore) sunt adesea folosite ca pseudo-spații, dar nu sunt permise alte semne de punctuație ciudate în numele unei variabile. (4)

Și întrucât suntem la subiect, permiteți-mi să vă încurajez să vă *numiți bine variabilele*. Aceasta înseamnă că fiecare nume de variabilă ar trebui să reflecte *exact* ceea ce reprezintă valoarea pe care o stochează. Exemplu: dacă o variabilă este menită să stocheze ratingul (în „stele”) pe care un utilizator IMDB l-a acordat unui film, nu o denumiți **movie**. Denumiți-o **rating**. (Sau chiar mai bine, **movie_rating**.) Credeți-mă: atunci când lucrezi la un program complex, există suficiente lucruri grele la care să te gândești fără să te încurci pe tine (și colegii tăi) prin nume de variabile apropiate, dar nu exacte. (5)

Acum amintiți-vă că o variabilă are trei lucruri – un nume, o valoare și un tip. Primele două apar în mod explicit în linia de cod în sine. În ceea ce privește tipul, de unde știe Python că **revolution** ar trebui să fie un „**int**?” Simplu: este un *număr fără punct zecimal*.

Ca o verificare a corectitudinii, putem cere lui Python să ne spună în mod explicit tipul variabilei, scriind acest cod:

```
type(revolution)
```

Dacă această linie de cod este executată după executarea celei anterioare, Python răspunde cu:

```
int
```

Deci, asta e.

Iată un alt „**fragment de cod**” (“code snippet” un termen care înseamnă doar „unele linii de cod pe care mă concentrez, care sunt, în general, doar o parte a unui program mai mare”):

```
revolution = 1776
moon_landing = 1969
revolution = 1917
```

Acum, dacă aceasta ar fi o clasă de matematică, acel set de ecuații ar fi absurd. Cum ar putea aceeași variabilă (revolution) să aibă două valori contradictorii? Dar într-un program, acest lucru este perfect legitim: înseamnă doar că imediat după executarea primei linii de cod, revolution

are valoarea **1776**, iar câteva momente mai târziu, după executarea celei de-a treia linii, valoarea sa s-a schimbat la **1917**. Valoarea sa depinde în întregime de „pe unde este programul” în timpul execuției sale.

Numere reale (fracționale): float

Singurul lucru ciudat despre al doilea tip de date din Python este numele său. În alt univers, s-ar fi putut numi o variabilă „reală” sau „zecimală” sau „fracționată”, dar din anumite motive istorice bizare este numită **float**. (6)

Toate aceleași reguli și reglementări se referă la **float**, la fel ca la **int**; singura diferență este că tastați un punct zecimal. Astfel:

```
GPA =3.17
price_of_Christian_Louboutin_shoes = 895.95
interest_rate = 6.
```

Rețineți că variabila interest_rate este într-adevăr un tip **float** (chiar dacă nu are parte fracțională) deoarece am tastat o perioadă:

```
type(interest_rate)
```

```
float
```

Text: str

Vorbind de nume ciudate, o variabilă de text Python este de tip **str**, care înseamnă „**șir**” (”**string**”. Ai putea să te gândești la asta ca la o grămadă de litere „strânse” împreună ca un colier cu mărgele.

Important: atunci când specificați o valoare **str**, trebuie să utilizați **ghilimele** (simple sau duble). În primul rând, așa știe Python că intenționați să creați un **str**, spre deosebire de un alt tip. Exemple:

```
slang = ‘lit’
grade = “3rd”
donut_store = “Paul’s Bakery”
url = ‘http://umweagles.com’
```

Observați, apropo, că un *șir de cifre* nu este același lucru cu un întreg. Pentru a înțelege:

```
schwarzenegger_weight = 249
action_movie = “300”
```

```
type(schwarzenegger_weight)
```

```
int
```

```
type(action_movie)
```

```
str
```

Vedeți? Ghilimelele fac toată diferența.

Lungimea unui șir

Probabil cel mai de bază este pur și simplu să întrebi despre **lungimea** unui șir sau despre numărul de caractere pe care le conține. Pentru a face acest lucru, includem numele variabilei între paranteze după cuvântul **len**:

```
len(slang)
```

```
3
```

```
len(donut_store)
```

```
13
```

După cum vom vedea, operația **len()** (și multe altele ca aceasta) este un exemplu de funcție în Python. În limbajul adecvat, atunci când scriem o linie de cod precum **len (donut_store)** spunem că „**apelăm funcția**”, ceea ce înseamnă pur și simplu să o invocăm sau să o declanșăm.

Mai mult lingo: din motive obscure, valoarea din interiorul parantezelor (aici, **donut_store**) este numită un **argument** pentru funcție. Și spunem că „**trecem**” unul sau mai multe argumente unei funcții atunci când o apelăm.

Toți acești termeni pot părea pedanți, dar sunt exacți și folosiți universal, așa că asigurați-vă că îi învățați. Linia de cod precedentă poate fi complet rezumată spunând:

„**Apelăm** la **funcția len()** și îi **trecem variabila donut_store** ca **argument**.”

Rețineți, apropo, că funcția **len()** așteaptă un argument **str**. Nu puteți apela **len()** cu o variabilă **int** sau **float** ca argument:

```
schwarzenegger_weight = 249
```

```
len(schwarzenegger_weight)
```

```
TypeError: object of type ‘int’ has no len()
```

(Ați putea crede că „lungimea” unui **int** ar fi numărul său de cifre, dar nu e așa.)

Un lucru pe care elevii îl confundă adesea este diferența dintre o *variabilă* de șir numită și cea a unei *valori* de șir (nenumită). Luați în considerare diferența în rezultatele următoarelor:

```
slang = 'lit'
len(slang)
```

```
3
```

```
len('slang' )
```

```
5
```

În primul exemplu, am întrebat „cât de lungă este valoarea în variabila **slang**?” Răspunsul a fost **3**, deoarece „**lit**” are trei caractere. În al doilea exemplu, am întrebat „cât de lung este cuvântul „**slang**”?” iar răspunsul este **5**. Nu uitați: numele variabilelor nu se pun niciodată între ghilimele. Dacă ceva este între ghilimele, este luat *literal*.

Combinarea și tipărirea variabilelor

Puteți face o mulțime de lucruri cu alte variabile decât să le creați. Un lucru pe care doriți să-l faceți frecvent este să **imprimați** o variabilă, ceea ce înseamnă să puneți valoarea acesteia pe pagină, astfel încât să o puteți vedea. Acest lucru se face cu ușurință apelând funcția **print()**:

```
print(donut_store)
print(price_of_Christian_Louboutin_shoes)
print("slang")
print(slang)
```

```
Paul's Bakery
895.95
slang
lit
```

Din nou, nu ratați diferența crucială dintre tipărirea lui „**slang**” și tipărirea lui **slang**. Primul este literal și cel de-al doilea nu. În prima dintre acestea, trecem cuvântul „**slang**” ca argument, nu variabila **slang**.

Deseori vom dori să combinăm biți de informații într-o singură instrucțiune de tipărire. De obicei, una dintre variabile este un șir care conține mesajul general. Există mai multe modalități de a realiza acest lucru, dar cea mai flexibilă se va dovedi a fi **metoda .format()**.

O **metodă** este foarte asemănătoare cu o funcție, dar nu exact. Diferența constă în sintaxa utilizată pentru a o numi. Când apelați o funcție (cum ar fi **type()** sau **len()**), pur și simplu introduceți numele acesteia, urmată de o pereche de paranteze în interiorul cărora puneți argumentele (separate prin virgule, dacă există mai multe). Dar când „apelați o metodă”, puneți o *variabilă* înainte de un punct („.”) și numele metodei, apoi parantezele. Aceasta este denumită „apelarea metodei **pe** variabilă”.

Sună mai confuz decât este. Iată un exemplu de **.format()** în acțiune:

```
price_of_Christian_Louboutin_shoes = 895.95
message = "Honey, I spent ${} today!"
print(message.format(price_of_Christian_Louboutin_shoes))
```

Rețineți cum scriem „**message.format**” în loc de „**format**”. Acest lucru se datorează faptului că **.format()** este o metodă, nu o funcție. Spunem că apelăm **.format()** „pe” **message** și trecem **price_of_ Christian_Louboutin_shoes** ca argument. (7) De asemenea, asigurați-vă că observați parantezele *duble* „))” la sfârșitul ultimei linii. Avem nevoie de amândouă, deoarece în programare, fiecare paranteză stângă trebuie să se potrivească cu o paranteză dreaptă corespunzătoare. Întrucât apelăm două funcții / metode pe o singură linie (**print()** și **.format()**), aveam două paranteze din stânga pe acea linie. Fiecare are nevoie de un partener.

În ceea ce privește specificul modului în care funcționează **.format()**, veți vedea că variabila de șir pe care o apelați poate include perechi de curlie (acolade). Acestea sunt substituenți pentru locul în care trebuie să lipiți valorile altor variabile în ieșire. Aceste variabile sunt apoi incluse ca argumente pentru metoda **.format()**. Codul de mai sus produce această ieșire:

```
Honey, I spent $895.95 today!
```

Adesea, în loc să creăm un nou nume de variabilă pentru a menține șirul preformatat, doar vom folosi **print()** literalmente, astfel:

```
print("Honey, I spent ${} today!".format(
    price_of_Christian_Louboutin_shoes))
```

Încă apelăm de fapt **.format()** la o variabilă aici, doar că nu ne-am deranjat să numim variabila. De asemenea, observați că codul nostru era prea lung pentru a se potrivi frumos pe o singură linie, așa că l-am rupt în două și am indentat a doua linie pentru a clarifica faptul că „**price_of_…**” nu își începea propria linie nouă. În mod crucial, toate parantezele sunt încă împerecheate, două câte două, chiar dacă parantezele din stânga sunt pe o linie diferită de parantezele din dreapta corespunzătoare.

În cele din urmă, iată un exemplu mai lung cu mai multe variabile:

```
name = "Pedro Pascal"
num_items = 3
cost = 91.73
print("Customer {} bought {} items worth ${}.".format(name,
    num_items, cost))
```

Customer Pedro Pascal bought 3 items worth $91.73.

Puteți vedea cum putem trece mai mult de un argument unei funcții / metode pur și simplu separându-le cu virgule în interiorul parantezelor.

Note

(1) Această utilizare a termenului „mediu" este diferită de termenul „mediu de programare".

(2) Strict vorbind, deși în limbaje precum Java variabilele au într-adevăr tipuri, în Python valorile au tipuri, nu variabilele. Această distincție nu este însă importantă pentru noi.

(3) Apropo, cuvântul cod este gramatical un substantiv de masă, nu un substantiv de numărare. Prin urmare, este potrivit să spunem „Am scris un cod aseară", nu „Am scris câteva coduri aseară". Dacă folosiți greșit acest lucru, veți fi marcat imediat ca începător.

(4) Oh, și o altă regulă: un nume variabil nu poate începe cu o cifră. Deci, **r2d2** este un nume de variabilă legală, dar nu și **007bond**.

(5) Și consider în totalitate faptul că variabila **revolution** nu este numită foarte bine. Am ales-o doar pentru a evidenția rapid o diferență.

(6) Dacă sunteți curios, acest lucru se datorează faptului că în limbajul de programare al computerului un „număr cu virgulă mobilă" înseamnă un număr în care punctul zecimal ar putea fi oriunde. Cu un număr întreg ca -52, punctul zecimal este implicit în partea dreaptă a secvenței de cifre. Dar cu numere precum -5,2 sau -,52 sau -000052 sau chiar 520000, punctul zecimal a „plutit" departe de această poziție fixă.

(7) Ori de câte ori mă refer la o metodă aici, voi pune un punct înaintea numelui său. De exemplu, nu este metoda „**format()**", ci metoda „**.format()**".